P5214/SMQ-042

United States Application
Entitled: A Flushable Free Register List

Inventors: Spencer Gold, Julie M. Staraitis, and Jason Eisenberg

A FLUSHABLE FREE REGISTER LIST

Technical Field of the Invention

The present invention generally relates to microprocessors and more particularly, to a method for managing physical registers.

Background on the Invention

High performance microprocessors, may use out-of-order instruction execution rather than conventional sequential execution that requires all instructions to execute in sequential order. As such, if the next sequential instruction does not have all of its operands in a valid state, the instruction pipeline stalls until the operands become valid. In contrast, an out-of-order microprocessor executes instructions as soon as its operands become valid, independent of the original instruction sequence. Consequently, as these high performance microprocessors execute out-of-order instructions, the microprocessors generate numerous temporary register results. The temporary values are stored together with permanent values in register files. The temporary values become permanent values when the corresponding instructions are retired. An instruction is retired when the temporary result becomes the new state of the microprocessor.

To ensure that each instruction is provided with the correct operand value, each logical register number referenced in the instruction is mapped to a physical register. Each time a new value is placed in a logical register, the logical register is assigned to a

new physical register. As a result, each physical register holds a single permanent or temporary value. In this manner, data dependency issues commonly associated with out-of-order instruction execution are avoided.

Since these high performance microprocessors perform out-of-order execution, an instruction can change its register value before all of the prior instructions complete. However, if any of the prior instructions cause an exception, all of the sequential instructions prior to the time the exception occurred are flushed. As a result, the registers allocated to the instructions being flushed become available for allocation to newly decoded instructions.

Typically, a free register list contains just enough capacity to track the maximum allowable number of free physical registers. When physical registers are assigned to an instruction a vacancy is created in the free register list that is immediately filled with a pointer to the physical register that will become free once that instruction retires. Consequently, the typical free register list does not store pointers to physical registers that were recently assigned to an incoming instruction. Hence, when it becomes necessary to process a flush, it is the contents of the free register list that must be restored to a prior state, not the registers in the physical register file. Unfortunately, this method is burdensome and time consuming because it requires at least two additional operations to restore a free physical register list to its previous state.

Summary of the Invention

The illustrative embodiment of the present invention addresses the above-described limitations of the conventional methods for restoring a free physical register

list to a previous state. The illustrative embodiment provides an approach that allows an instruction pipeline flush to occur without having to physically restore register values. Moreover, the illustrative embodiment prevents the physical register pointers pointing to the physical registers that are temporarily or speculatively assigned as a destination operand for a instruction from being overwritten with physical register pointers to registers that will soon become free.

In a first embodiment of the present invention, a method for register allocation is performed in a microprocessor performing speculative instruction execution. A data structure that includes a free physical register list to track physical register allocation for an independent program execution stream or a thread of the microprocessor is provided. The free physical register includes a set of managing pointers that include a read row pointer, a write row pointer, and a retire pointer. The managing pointers operate to identify the locations of physical registers available for assignment as designation registers, the location of physical registers speculatively allocated as destination registers for a set of instructions, and the locations of physical registers allocated to instructions that are about to be retired. As such, the speculatively allocated physical registers for a speculative instruction set can be restored to their previous state when the microprocessor issues a flush operation by moving the read row pointer to the first speculatively allocated physical register for the flushed instruction bundle.

In another embodiment of the present invention, a method for register allocation is performed in a multithreading microprocessor performing speculative instruction execution. A data structure that includes a free physical register list for each thread of the multithreading microprocessor that independently track physical register allocation

for a first thread and independently track physical register allocation for a second thread is provided. The free physical register list for the first thread includes a first set of managing pointers that include a read row pointer, a write row pointer, and a retire pointer. In similar fashion, a second set of managing pointers that include a read row pointer, a write row pointer, and retire row pointer manage physical register allocation for the second thread. The managing pointer sets are independent of one another and therefore independently track the physical registers available for assignment as designation operands for the first thread and for the second thread of the multithreading microprocessors. The managing pointers operate to identify the locations of physical registers available for assignment as designation registers, the location of physical registers speculatively allocated as destination registers for a set of instructions, and the locations of physical registers allocated to instructions that are about to be retired. As such, the speculatively allocated physical registers for a speculative set of instructions can be restored to their previous state when the multithreading microprocessor issues a flush operation by moving the read row pointer associated with the thread being flushed to the first speculatively allocated physical register for the flushed instruction set.

The above described approach benefits a microprocessor architecture that processes speculative data because the physical register list is quickly restored to a previous state without having to physically restore any data. In addition, the use of two sets of row pointers to manage register pointers allows an instruction bundle from one thread of the microprocessor to be flushed without disturbing operation of other threads of the microprocessor. Moreover, each managing pointer set is structured to prevent the speculatively allocated physical register pointer from being overwritten with a physical register pointer allocated to an instruction that is about to be retired.

In accordance with another aspect of the present invention, a semiconductor device having multiple physical registers that are assigned as destination operands for instructions to be executed by a microprocessor includes a module that provides a structure for holding information that identifies the physical registers that are available for holding the destination operand. The information identifying the available physical registers is managed by a set of row pointers. The set of row pointers identify the available physical registers for a thread of a microprocessor. The set of managing row pointers includes a read pointer, a write pointer and a retire pointer. In the pointer set, the read pointer and the write pointer are at a fixed distance from each other and traverse the data structure in unison. Moreover, the starting location of the physical registers speculatively assigned as destination operands for a set of instructions is determined by subtracting from the position of the retire pointer the fixed distance between the read pointer and the write pointer while the ending location of the physical registers speculatively assigned as an operand destination for an instruction set is determined by the read pointer.

In accordance with yet another aspect of the present invention, a semiconductor device having multiple physical registers that are assigned as destination operands for instructions to be executed by a microprocessor includes a module that provides a structure for holding information that identifies the physical registers that are available for holding the destination operands. The information identifying the available physical registers is managed by two sets of row pointers that operate independently of one another. The first set of row pointers identify the available physical registers for a first thread of a multithreading microprocessor and the second set of pointers identifies the

available physical registers for a second thread of the multithreading microprocessor. Each set of managing row pointers includes a read pointer, a write pointer and a retire pointer. In each pointer set, the read pointer and the write pointer are at a fixed distance from each other and traverse the data structure in unison. Moreover, the starting location of the physical registers speculatively assigned as destination operands for a set of instructions is determined by subtracting from the position of the retire pointer the fixed distance between the read pointer and the write pointer while the ending location of the physical registers speculatively assigned as an operand destination for a set of instructions is determined by the read pointer.

In accordance with yet another embodiment of the present invention, a computer readable medium holding computer executable instructions provides a method for tracking physical registers in a structure holding information identifying physical registers that are free to be assigned as destination operands for instructions executing on a microprocessor. The destination operand identifies where data resulting from an executed instruction is to be stored. The method allows a set of pointers to manage the physical registers for a thread of the microprocessor. The set of pointers includes a read pointer to indicates a free or available register, a write pointer that indicates where an assigned physical register corresponding to a soon-to-be retired instruction should be written and a retire pointer that indicates a physical register corresponding to an instruction that is the next instruction to be retired. Should the read pointer advance to within a predetermined number rows of the retire pointer, a stall condition may be initiated to prevent the assignment of registers that are not yet free. The stall condition persists until the retire pointer is moved a sufficient distance away from the read pointer to enable further assignment of free registers. Moreover, the state of the structure may

be restored when an instruction pipeline flush occurs by resetting the read pointer to point to the physical register allocated to the just flushed instruction

Brief Description of the Drawings

An illustrative embodiment of the present invention will be described below relative to the following drawings.

Figure 1 is a block diagram illustrating a microprocessor suitable for practicing the illustrative embodiment of the present invention.

Figure 2 depicts a physical register list that is suitable for practicing the illustrative embodiment of the present invention.

Figure 3 is a flow chart illustrating the steps taken to allocate free physical registers.

Figure 4 is a flow chart illustrating the steps taken to process a flush request.

Detailed Description

The illustrative embodiment of the present invention provides a method and a semiconductor device for increasing the efficiency of restoring a free physical register list to its preexisting state following an instruction flush request. In the illustrative embodiment, the physical register list stores and maintains the pointers to the multiple

physical integer registers that are currently available for use as destination registers for a newly fetched instruction of a multi threaded microprocessor. The physical register list stores and maintains this information for the threads of the microprocessor. The physical register list may be restored to a preexisting state following a flush request by independently resetting a read row pointer to the physical register pointer assigned to the flushed instruction. As a result, temporary or speculatively assigned physical registers will be immediately reused by newly incoming instructions following a flush request, which enables more efficient use of the processor's physical register resources. Furthermore, the free register list is organized such that the prior state of the free register list necessary for correctly processing a flush request is certain to remain within the free register list, which eliminates the need for restoring this state from a separate memory device.

The ability to prevent the overwriting of physical register pointers that have been speculatively assigned to incoming instructions ensures that the free register list can be restored to its previous state. The free physical register list of the illustrative embodiment utilizes twice as many rows as the maximum number of free physical registers. In this manner, the read pointer and the write row pointer are able to be set apart at a fixed distance and moved in unison to prevent overwriting of pointers to physical registers that are allocated to instructions that have not yet retired.

In the illustrative embodiment, the method and semiconductor device are attractive for use for simultaneous multithreading processors. As a result, processor performance is increased by avoiding having to physically reset the free physical register pointer values to their previous state upon an instruction pipeline flush. Thus, pointers

to all physical registers allocated to instructions that have not yet retired are protected from being overwritten until the instructions commit and the temporary results are retired as the new state of the processor.

Figure 1 illustrates a microprocessor suitable for practicing the illustrative embodiment. The microprocessor 10 includes a mapping table 12, a physical register list 14, and a memory array 16. The microprocessor 10 is adapted for simultaneous multithreading and may execute instructions out-of-order.

The mapping table 12 operates to map a logical register to a physical register when the instruction is being decoded. The physical register list 14 stores and maintains pointers to the physical integer registers that are currently available or soon to be available for use as a destination register of a just fetched instruction. The microprocessor 10 in the illustrative embodiment is a dual threaded processor. Nevertheless, those skilled in the art will appreciate that the microprocessor need not be a dual-threaded microprocessor, but rather maybe a microprocessor having more than two threads. The physical register list stores and maintains pointers to physical integer registers for both threads. Nevertheless, those skilled in the art will recognize that the physical register list 14 may store and maintain pointers to physical integer registers for a single-thread of a single threaded microprocessor.

Since the microprocessor 10 is capable of out-of-order instruction execution, the microprocessor 10 generates numerous speculative or temporary register results. These speculative values are stored in a physical register together with permanent values from previously retired instructions. An instruction is considered retired when the temporary

value of the physical register is committed as the new state of the processor. An instruction can retire only after all previous instructions have been successfully retired.

To ensure that each decoded instruction is given the correct operand value, the logical register number used in the instruction is mapped by the mapping table 12 to a physical register pointer in the physical register list 14. Each time a new value is placed in a logical register, a new physical register is assigned to the logical register by the mapping table 12. Consequently, each physical register holds only a single speculative value. Upon the instruction being retired, the new permanent value in the physical register is written to the memory array 16.

As illustrated in Figure 2, the physical register list 14 of the illustrative embodiment is organized as a modulo-eight (8), dual ported (one read and one write) register file structure with 128 rows and 16 columns. The modulo-eight (8) organization allows eight rows of data to be read in each read cycle even though the register file has a single read port. Those skilled in the art will recognize that the number of rows and columns of the physical register list can be extended to include additional rows and additional columns or can be reduced in size to include fewer than 128 rows and 16 columns.

The physical register list 14 tracks available physical registers for a first thread (thread 0) in a first portion 20 of the physical list register 14 and tracks the available physical registers for a second thread (thread 1) in a second portion 22 of the physical register list 14. Each portion 20 and 22 of the physical register list 14 include an

independent set of pointers to manage the allocation of free physical registers. The operation of each set of pointers is discussed below in more detail.

Read row pointer 36, write row pointer 38, and retire row pointer 40 function to manage the first portion 20 of the physical register list 14. In like fashion, read row pointer 30, write row pointer 32, and retire row pointer 34 manage the second portion 22 of the physical register list 14. Read row pointer 36 and write row pointer 38 are set apart at a fixed distance from one another and move up and down the first portion 20 in unison. The number of rows between the read row pointer 36 and the write row pointer 38 is determined by the number free or available physical integer register pointers that the physical register list 14 is designed to manage. Those skilled in the art will recognize that the number of available physical integer register locations in the physical register list 14 is driven by the need of the microprocessor 10. In the illustrative embodiment of the present invention, the read row pointer 30 and the write row pointer 38 are sixty-four (64) rows apart. The retire row pointer 40 travels independently of the read row pointer 36 and the write row pointer 38.

In similar fashion, the read row pointer 30 and write row pointer 32 of the second portion 22 of the physical register file 14 are also set apart by a fixed number of rows, and move up and down the file structure in unison. Likewise, the retire row pointer 34 moves independently of the read row pointer 30 and the write row pointer 32.

The physical register list 14 includes the read word line 44 to read data in a particular row of the first portion 20 or the second portion 22 of the structure. The physical register list 14 also includes the write word line 46 to write data into a

particular row of either the first portion 20 or the second portion 22 of the physical register 14. Moreover, because the physical register list 14 is implemented as a modulo-eight (8) structure, the physical register list 14 allows up to eight rows of data to be read in each read cycle although the physical register list 14 has a single read port 44. Thus, the physical register list 14 is capable of providing up to eight physical register pointers to the microprocessor 10 with a single read. Those skilled in the art will recognize that the physical register list 14 is capable of providing fewer than eight physical register pointers with a single read, for example one physical register pointer.

As illustrated, the read row pointer 30 and the read row pointer 36 each indicate the next available physical register pointer that are available for assignment as a destination operand of a logical register. The read row portion 30 corresponds to the second portion 22 of the structure and the read row pointer 36 corresponds to the first portion 20 of the structure. The write row pointer 32 and the write row pointer 38 each indicate where a physical register pointer should be written for a physical register holding an operand value corresponding to a logical register soon to be overwritten by an instruction in the pipeline and thus become a free register once that instruction is retired. The write row pointer 32 corresponds to the second portion 22 of the structure and the write row pointer 38 corresponds to the first portion 20 of the physical register list 14. The retire row pointer 34 and the retire row pointer 40 point to in each of their respective portions 22 and 20 of the physical register list 14, the physical register pointer of a physical register assigned to an instruction that is the next instruction to be retired by the microprocessor 10.

During initialization of the physical register list 14 the read row pointers 30, and 36 are set to row zero and the write row pointers 32 and 38 along with the retire row pointers 34, and 40 are set to row sixty-four of the physical register list 14. Thus, after initialization of the physical register list 14, the available physical registers are indicated by the physical register pointers located in row zero through row sixty-three of the first portion 20 and the second portion 22 of the physical register list 14.

The protected register region 24 of the second portion 22 holds values identifying the physical register pointers that are allocated to instructions having a logical destination register that have been decoded and issued but have not yet retired. The physical registers referenced to by the pointers in the protected region 24 hold the speculative or uncommitted data values generated by the instructions that have not yet retired. The available region 26 of the second thread portion 22 indicates the register pointers for the physical registers that are available to be allocated as a destination operand for instructions yet to be fetched and decoded by the microprocessor 10. The available region 26 is determined by the location of the read row pointer 30 and the location of the retire row pointer 34 along the first portion 22 of the physical register list 14. The waiting retirement region 28 of the first portion 22 indicates the physical register pointers of the physical registers corresponding to logical registers soon to be overwritten by instructions in the pipeline. Once these instructions are retired, the data values being stored in these registers become obsolete, and therefore become available registers. The row location of the retire row pointer 34 is sixty-four rows down from the first row in the protected region 24. The first portion 20 of the physical register list 14 also includes a protected region and a waiting retirement region although not illustrated

in Figure 2 because the first portion 20 illustrates the initialization state and the second portion 22 illustrates a working state of the physical register list 14.

With reference to Figure 2 and Figure 3, the utilization of the read row pointer 30, the write row pointer 32 and the retire row pointer 34 will be discussed below. In addition, Figure 2 and Figure 4 illustrate the processing of a flush request without having to physically restore any physical register values. For ease of the discussion below, the description of the pointer set utilization and the description of the flush operation will be discussed with reference to the second thread portion 22 of the physical register list 14. The first thread portion 20 and the second thread portion 22 of the physical register list 14 operate independently of one another to read, write, and retire physical register pointers and also operate independently to process a requested flush operation. As such, the discussion below will focus on the second thread portion 22. Those skilled in the art will recognize that in discussing the second thread portion 22 in relation to an instruction flush request is not meant to be limiting of the present invention, but meant to merely ease the explanation of the flush operation itself.

Upon initialization of the physical register list 14, the read row pointer 30 points to row zero and the write row pointer 32 and the retire row pointer 34 point to row sixty-four of the second threaded portion 22 of the physical register list 14 (step 50 in Figure 3). After an instruction is fetched and decoded that includes a logical destination register, the microprocessor 10 reads the physical register pointer that the read row pointer 30 points to and assigns the corresponding physical register to the logical register as a destination register to identify where data resulting from an operation is to be stored (step 52 in Figure 3). At about the same time, the pointer to the physical register last

allocated to the logical register of the just fetched and decoded instruction is written into the free register list at the location pointed to by the write pointer (step 52 in Figure 3). At this point, the read row pointer 30 and the write row pointer 32 are updated and moved down one row in the second thread portion 22 of the physical register list 14, and the just read physical register pointer becomes part of the protected region 24 (step 54 in Figure 3). The protected region 24 is defined as the rows between the read row pointer 30 and the row that is the fixed distance between the read and write pointers subtracted from the retire pointer on the other end. For example, in Figure 2, the protected region 24 includes rows six through nineteen of the first threaded portion 22.

In addition, in order for an out-of-order processor to support precise exceptions, it must retire instructions in the same order that they were fetched, i.e., in program order. Thus, the destination registers of instructions become available in the same order that they were assigned. As such, each time an instruction retires, the free register list moves the retire row pointer 34 down a row, which indicates that the contents of the physical register previously pointed to by the retire pointer are obsolete and can be safely overwritten.

If during the update the read row pointer 30 comes within a predetermined number of rows of the retire row pointer 34, a stall condition exists (step 56 in Figure 3). The stall condition exists when the read row pointer 30 gets too close to the retire row pointer 34 indicating that the available region 26 may be too small to provide an adequate number of destination registers for the next set of instructions. Consequently, the microprocessor 10 stalls, meaning that it stops fetching instructions until more registers are added to the available region 26. To cure the stall condition, the

microprocessor 10 allows instructions in the pipeline to retire, which advances the retire row pointer 34 towards the write row pointer 32 and away from the read row pointer 30 (step 58 in Figure 3). When enough instructions have retired to make the available region 26 sufficiently large, the microprocessor 10 may continue to fetch new instructions and the stall condition concludes. The predetermined distance typically corresponds to the number of instructions the microprocessor 10 can fetch in a single cycle. In the illustrative embodiment, the predetermined unsafe distance between the read row pointer 30 and the retire row pointer 34 is eight rows because the microprocessor 10 can fetch of up to eight instructions in a single cycle. If there is a sufficient number of rows between the read row pointer 30 and the retire row pointer 34 a stall condition does not exist and the physical register pointers in the available region 26 may be read and allocated as destination operand locations for logical registers (step 52 in Figure 3).

If during the execution of a set of instructions by the microprocessor 10, a prior instruction causes an exception or a branch misprediction, a flush request is issued to restore the physical register list to the state that existed when the flush target instruction was originally assigned a destination register (step 64 in Figure 4). If a prior instruction exception or a branch misprediction is encountered, the read row pointer 30 is reset by moving the read row pointer 30 to the first physical register pointer assigned to the first logical register in the instruction bundle that is about to be flushed (step 66 in Figure 4). Moreover, since the read row pointer 30 and the write row pointer 32 move in unison, the write row pointer 32 is also reset. Consequently, because the instruction never retired, the prior permanent destination operand was not overwritten by the speculatively or temporary destination operand and the physical register list 14 is returned to its prior

state without having to physically restore any data. If no flush request is issued (step 64 in Figure 4) the read row pointer 30 and the write row pointer 32 are updated to indicate the number of physical register pointers assigned to logical registers (step 68 in Figure 4).

Those skilled in the art will recognize that the microprocessor 10 can issue a flush request at any point during instruction execution and cause the read and write row pointers to be reset. Moreover those skilled in the art will appreciate that the physical registers indicated by the register pointers in the protected region 24 cannot be overwritten with a temporary value because if the read row pointer 30 is within the predefined limits of the protected region, the retire row pointer 34 and the write row pointer 32 are also within the same predefined limit and a stall condition exists. The stall condition may be resolved by retiring physical register pointers, that is making them available for assignment as a destination operand location for a logical register or a flush request may be processed to return the physical register list to its previous state.

While the present invention has been described with reference to a preferred embodiment thereof, one skilled in the art will appreciate that various changes in form and detail may be made without departing from the intended scope of the present invention as defined in the pending claims. For example, the physical register list can track more than two threads or the register file list may track thirty-two physical registers or may have thirty-two columns.